位运算

2018年12月11日 13:38

- 位

一. 位

1. m位二进制数中，最右的为最低位、第0位
2. 此时最高位为最左的m-1位
3. 一字节=8位=8bit
4. 0xFF是8个1，是单字节表示的最大正数255，或最大负数-1
5. 运算时，以运算数类型的最多位数保存临时运算结果，而并不care最终会赋值给什么类型的数据，所以有时要在等号后手动加一个(long long)
6. 异或1取反，异或0不变
7. 与0改0，与1不变
8. 或1改1，或0不变

二. 补码

1. 反码/一补数：每个位都取反
2. 补码/二补数：反码+=1
3. 因为有符号正数的最高位10视作正负号，所以空出一个0才能表示正数
4. 所以0x7F FF FF FF是带符号最大整数21 4748 3647
5. 为防止加法溢出，一般用memset出0x3f 3f 3f 3f=10 6110 9567

三. 移位

1. 左移：最低位补0，最高位舍弃
   i. m<<1相当于2m，相当于右边补一个0
   ii. 1<<n相当于2^n，相当于1后面n个0
   iii. (1<<n)-1相当于2^n -1，相当于n个1
   iv. m<<n相当于m* 2^n
2. 算术右移：最高位取符号位，最低位舍弃
   i. (-3)>>1 = -2
   ii. 3 >>1=1
   iii. m>>1相当于m/2向下取整
3. 逻辑右移：最高位补0，最低位舍弃
   i. 好像没什么意义?
   ii. 因为c++语法没有规定右移实现方式，说不定有的编译器会这么做……
4. eg：快速幂a^b%p和龟速乘a*b%p
   i. 思路：将b拆成2的幂级数的形式，再利用a^(2^i)=(a^(2^(i-1)))^2
   ii. 时间复杂度：O（log2b）

```
int qpow(ll a,int b,int p){
     ll ans=1;
     for(;b;b>>=1,a=a*a%p)
          if(b&1) ans=ans*a%p;
return ans;}
ll smul(ll a,ll b,ll p){  //记得传参时先给ab取余一发p
     ll ans=0;
```

```
        for(;b;b>>=1,a= (a<<1)%p)
            if(b&1) ans= (ans+a)%p;
    return ans;}
```

四. 二进制状态压缩

1. c艹的运算符优先级：加减、**位移、大小、位逻辑、逻辑**、赋值
    i. 即：位移、大小等、等不等、位且、位异或、位或、且、或
2. n & 1：取最右第0位（之后也都是从0开始数的）
3. (n>>k) & 1：取右第k位
4. n & ((1<<k)-1)：取右k位
5. x ^ (1<<k)：右第k位取反
6. n | (1<<k)：右第k位改1
7. n & (~(1<<k))：右第k位改0
8. eg：枚举最短哈密顿路径
    i. 思路：brute-forth枚举，f[i][j]i是二进制数据表示经过某些点与否，j表示正在找哪个点的最小权
    ii. 时间复杂度：由状态压缩dp从O（n*n！）优化至O（n方*2^n）
    iii. int dp[1<<20][20];

```
int hamilton(int n,int weight[20][20]){
    memset(f,0x3f,sizeof(f));f[1][0]=0;//第0点作起点for_(i,1,1<<n)//举完所有走过与没走过的情况
        for_(j,0,n) if(i>>j&1)//逐个判断i情况走过的所有点j
            for_(k,0,n) if((i^1>>j)>>k&1)
//i情况第j点取0后，第k点是否走过。即这次k遍历只找走过的点
                f[i][j]=min(f[i][j],f[i^1<<j][k]+weight[k][j]);
//如果i情况的j从k点走的权更小，就更新
    return f[(1<<n)-1][n-1];}
```

五. 成对变换

1. 非负偶数n^1得n+1
2. 非负奇数n^1得n-1
3. 可用于存储无向边或双向的平行边

六. lowbit运算

1. lowbit(n)意为将二进制n的最低位1左边的数全赋为0得到的值
2. lowbit(n)=n&(-n)=n&(~n+1)
    i. 其中：补码-n=~n+1
3. 快速拆分正整数为二幂和形式
    i. while(n){
        cout<<(n&-n)<<endl;
        n-=n&-n;}
    ii. 事先初始化好map<ull,int>m;m[1<<i]=i;就可以快速得知从右往左第几位是1（从0开始数）
    iii. 如果是int范围的数据，可以利用2^k mod 37在k<35时互不相同，而用数组m[(n&-n)%37]代替map映射

七. 例题

1. 手写bitset
   i. 建立大小为N的bitset：int v[N/32+5];
   ii. 在n处置1：v[n>>5]|=1<<(n&31);
   iii. 求n处的值：return v[n>>5]&(1<<(n&31));
2. 经一顿左移或右移后将序列变成任一相同数，所需的做少移动次数
   i. 初始情况及右移吞掉了1的情况 iff 左移会出现新的数，需要特判
   ii. 记录数字范围内每个数转移次数在ans里，记录能转成它的数的数量在cnt里，cnt==n时更新o
   iii.

```
for_(i,0,n){
        scanf("%d",a+i);
        rof_(j,19,0)
                if(a[i]>>j & 1){
                        fst1[i]=j;          //a[i]>>fst1[i]得到a[i]最左的1
                        ++ cnt[0];
                        ans[0]+= j+1;
                        break;}}
int M= 1<<19;
for_(i,0,n){       //先看看不动和左移能移出什么
        int t= a[i];
        int tt= 0;
        while(t<M){
                ++cnt[t];
                ans[t]+= tt;
                ++tt;
                t<<= 1;}}
for_(i,0,n){
        for__(j,1,fst1[i]){
                bool flag= a[i]>> (j-1) & 1;
                int t= a[i]>> j;
                ++ cnt[t];
                ans[t]+= j;
                if(t && flag){     //本次右移删去了一个1，需更新右移结果
                        int tt= j;
                        while(t && t<M){          //t是0或t太大了
                                t<<= 1;
                                ++cnt[t];
                                ++tt;
                                ans[t]+= tt;}}}}
int o= 1<<30;
for__(i,0,M){
//              cout<<i<<"  "<<cnt[i]<<"  "<<ans[i]<<endl;
        if(cnt[i] == n)
                o= min(o, ans[i]);}
cout<<o;
```